Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

# AnoMLP-TCP: An Anomaly Aware Multilayer Perceptron for Test Case Prioritization in Continuous Integration

**MUSA ISYAKU**

musaisyaku77@gmail.com

Abubakar Tafawa Balewa University https://orcid.org/0009-0008-8701-1090

**Research Article**

**Keywords:** Test case prioritization, continuous integration, regression testing, multilayer perceptron, isolation forest, anomaly detection

**Posted Date:** October 7th, 2026

**DOI:** https://doi.org/10.21203/rs.3.rs-11266283/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

**Additional Declarations:** The authors declare no competing interests.

# AnoMLP-TCP: An Anomaly Aware Multilayer Perceptron for Test Case Prioritization in Continuous Integration

**Musa Isyaku*, Abdulsalam Ya'u Gital**, Haruna Chiroma*****
Faculty of Computing, Abubakar Tafawa Balewa University, Bauchi, Nigeria

*Corresponding author: musaisyaku77@gmail.com

## Abstract

Continuous integration (CI) has made regression testing more frequent. However, the growing size of automated test suites can delay feedback to developers. We propose AnoMLP-TCP, a lightweight anomaly aware multilayer perceptron (MLP), as an alternative to AnoLSTM-TCP for test case prioritization. AnoMLP-TCP is based on the MLP architecture and uses nine inputs derived from historical execution data, including execution duration, time since the last execution, historical failure count, historical pass count, changes in verdict status, the three most recent verdicts, and an isolation forest anomaly score. Our study uses a chronological cycle-based split with the most recent 20% CI and evaluates our model on three ABB Robotics Norway datasets: IOFROL, PaintControl and ABB. Our results show that AnoMLP-TCP achieved APFD values of 0.821, 0.959 and 0.893, respectively. AnoMLP-TCP exhibited the same per-cycle APFD values as AnoLSTM-TCP on the examined cycles, resulting in zero paired differences and preventing estimation of the Wilcoxon signed-rank test for this comparison. AnoMLP-TCP has a total of 13,057 parameters, compared with 54,577 for AnoLSTM-TCP. This means that there is a 76.1% decrease in the total number of parameters and a lower training time per epoch and prioritization time. At 50%, 70%, 90%, and 100% test-selection budgets, AnoMLP-TCP and AnoLSTM-TCP also produced identical NAPFD values. Overall, the results show that AnoMLP-TCP achieves APFD performance comparable to that of the reproduced AnoLSTM-TCP baseline while substantially reducing model complexity and training cost per epoch. The study further evaluates predictive error measures, fault-detection timing, and performance under partial test-execution budgets.

**Keywords:** Test case prioritization, continuous integration, regression testing, multilayer perceptron, isolation forest, anomaly detection.

## 1. Introduction

Software testing demonstrates that a system continues to meet its requirements after changes to the code are made. Continuous integration (CI) drives changes to be integrated and tested frequently, producing valuable feedback. However, it is often difficult to run the entire test suite within the CI budget, creating a bottleneck. Test case prioritization (TCP) can help alleviate this bottleneck without creating a new test suite or removing tests from the existing one. Rather than altering test suite membership, TCP reorders the existing suite to place tests viewed most likely to fail before those less likely to fail. This can increase the chances of finding faults early in a test run.

The existing methods include traditional fault-based test case prioritization methods. Traditional prioritization methods use information from historical failures, execution duration, coverage, and code changes. Machine learning methods extend the traditional approach by learning the relationships between the historical behavior patterns of tests and test outcomes. When these learning models are used, the models can prioritize tests for future runs. Recent studies have developed prioritization methods that use deep learning architectures, namely, recurrent neural networks. Deep learning models can model more sophisticated patterns in the execution history. However, recent studies have shown that certain prior methods are as effective as state-of-the-art deep learning methods when they are assessed against long-term and industrial CI test suites. This study extends the analysis with additional datasets and tests a reinforcement-learning-based method. The evaluated reinforcement-learning-based method produced lower APFD values than several of the other evaluated approaches under the experimental protocol. (Choudhury et al. 2025) proposed a new anomaly aware test case prioritization method called AnoLSTM-TCP, which adapts an LSTM-based sequential learning model, depending on an isolation forest anomaly detection score. The AnoLSTM-TCP method was developed and tested on the IOFROL dataset and the PaintControl dataset from ABB Robotics Norway.

Replacing the RNN-based recurrent LSTM with a simple MLP decreases complexity but provides similar prioritization efficacy. We propose the AnoMLP-TCP model, which replaces the LSTM component of AnoLSTM-TCP with a compact feedforward MLP while retaining the anomaly aware feature representation. The proposed model is evaluated and compared with AnoLSTM-TCP in terms of modeling and predictive error measures. The studies involve the effects of repurposed data-driven representations of test suite execution captured with a time series LSTM model. AnoMLP-TCP is also compared with the regression-based AnoLSTM-TCP on the basis of the metric of APFD. AnoMLP-TCP achieves APFD performance comparable to that of the reproduced AnoLSTM-TCP baseline while substantially reducing model complexity and training cost per epoch. The study makes four contributions. First, an anomaly aware feedforward TCP model based on the feature representation used by AnoLSTM-TCP is proposed. Second, it evaluates the model with a chronological, cycle-based split that better reflects future CI deployment. Third, it compares prioritization effectiveness, predictive error measures, fault-detection timing, and computational cost. Fourth, it evaluates whether the observed equivalence with AnoLSTM-TCP remains when only part of a test suite can be executed.

Section 2 of this study discusses the Background of the studies; Section 3 discusses the related work on TCP and places the study within the literature. Section 4 presents the AnoMLP-TCP framework and provides details on the data sources, preprocessing, feature engineering, anomaly detection, evaluation protocol, and architecture of the AnoMLP-TCP model. Section 5 describes the experimental setup and evaluation metrics. Section 6 reports and discusses the results. Section 7 discusses research threats to validity, and Section 8 concludes the study and proposes directions for future research.

## 2 Background

### 2.1 Test case prioritization

Test case prioritization (TCP) is a regression testing optimization technique that ranks test cases by their potential to reveal faults early, ensuring that high-priority tests are executed first within a limited available time. The TCP techniques can generally be broken down into three categories: white-box, black-box, and hybrid approaches. White-box techniques dig into internal details, such as the code structure and coverage, whereas black-box techniques focus mainly on functional aspects, execution history, or test results. On the other hand, hybrid techniques blend both structural insights and execution-based information for a more comprehensive approach. It is also known as *glass box testing*, as everything that is required to implement the software is visible. The black-box technique is one of the major techniques in dynamic testing for designing effective test cases on the basis of functional requirements and execution history, and hybrid techniques integrate both *white-box* (structural/code-based) and *black-box* (execution/data-driven) elements to increase prioritization accuracy. In CI environments, black-box and hybrid approaches are most attractive when source-code instrumentation is unavailable or undesirable.

The effectiveness of a prioritization strategy is typically measured via the average percentage of faults detected (APFD) metric, which quantifies how quickly faults are detected across the ordered test execution sequence. A higher APFD value indicates that failing test cases are ranked earlier, resulting in faster fault detection.

### 2.2 Multilayer perceptron

The MLP is a type of artificial neural network that consists of multiple layers of interconnected neurons, enabling it to learn complex patterns and relationships within data. MLPs are particularly well suited for tasks involving classification and regression, making them valuable in various applications, including test case prioritization. An MLP typically comprises an input layer, one or more hidden layers, and an output layer. Each neuron in the network processes input data and applies an activation function to determine its output. The connections between neurons are associated with weights, which are adjusted during the training process to minimize the difference between the predicted and actual outcomes.

In the context of test case prioritization, MLPs can effectively capture the nonlinear relationships among various test case attributes, such as complexity, execution time, and historical defect data. By training on historical test case information, MLPs can learn to identify which test cases are more likely to reveal defects, allowing for more informed prioritization decisions. The ability of MLPs to model complex interactions makes them powerful tools for improving test case prioritization. By leveraging the strengths of machine learning, MLPs can increase fault detection rates, improve resource allocation, and ultimately lead to more efficient testing processes in continuous integration environments. As software

systems continue to evolve, the application of MLPs in test case prioritization represents a promising avenue for advancing software quality assurance practices. The structure of the MLP is shown in Figure 1 below.

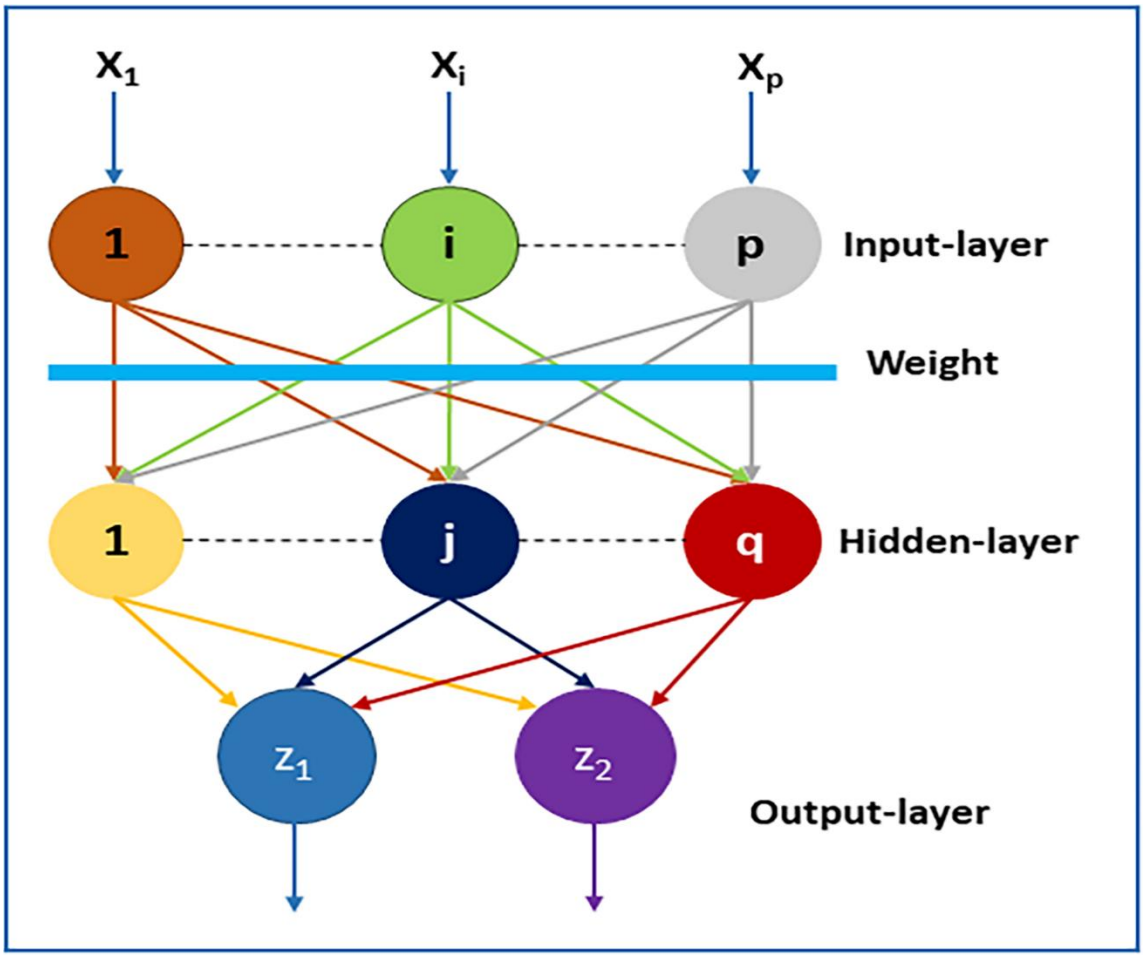



Figure 1. Structure of a multilayer perceptron (Mustaqeem et al. 2024)

## 3 Related Work

### 3.1 Deep learning and anomaly-based approaches

(Choudhury et al., 2025) proposed AnoLSTM-TCP, an anomaly based long short-term memory method for TCP, which achieves APFD values of 0.85 and 0.90 on the IOFROL and PaintControl datasets, respectively, outperforming RETECS, DeepGini, and DeepOrder. While AnoLSTM-TCP demonstrates the power of temporal modeling for TCP, it does not provide a systematic comparison of classical ML models under identical experimental conditions, leaving the relative contribution of deep learning over simpler baselines unclear. (Romero et al., 2026) used grammar-guided genetic programming to automatically select and tune ML pipelines for test-case prioritization. The approach adapted to changing software builds but does not examine lightweight neural architectures. (Khan et al., 2025) developed an ML-based TCP framework for CI via the IBM Open Liberty dataset. Prioritized builds achieved approximately 50% higher mean APFD, but the study did not focus on lightweight or anomaly aware neural models. (Behera & Abhinna, 2025) developed a GRU-based deep learning algorithm to prioritize test cases by applying historical test information. The algorithm was tested on the PaintControl and IOFROL datasets via the APFD, MSE, MAE, RMSE, $R^2$ and time indicators. The results obtained are APFD values of 0.77 and 0.70 and total execution times of 0.02 s and 0.01 s, respectively. The weaknesses of this approach include testing the model on only two datasets and possible overfitting or unstable training. (Kalyani et al., 2025) suggested BugPrioritizeAI, which is a multimodal deep learning architecture for test case prioritization. This approach integrates bug reports, changes to the source code, and tests metadata via BugTestRankNet and attention-based fusion. The APFD was 84.3%, which is higher than that of DeepTCP (77.9%) and RiskMiner (74.5%). However, it is based on correct metadata and has been tested mostly on Java-based open-source projects.

(Sharif et al., 2021) proposed Deeporder, a deep learning-based prioritization approach evaluated on ABB and related datasets. (Medhat et al., 2020) combined deep learning with search-based techniques for CI regression testing in IoT systems. (Shankar & Sridhar, 2024) applied deep reinforcement learning for TCP, whereas (Lousada & Ribeiro, 2020) explored neural network embeddings. These methods, while innovative, are often evaluated on single datasets with limited cross-dataset generalizability.

### 3.2 Classical machine learning approaches

Prior work on TCP has focused on deterministic heuristics that incorporate execution cost, failure history, coverage, and domain-specific knowledge. RETECS [9] (Spieker et al., 2017)was the first to frame TCP as an RL problem, which allows a policy to benefit from multiple CI executions. The main contribution of the RETECS paper is that, ultimately, the value of prioritization is best expressed in terms of the speed at which useful fault information is obtained; accuracy is not the prime consideration. This insight guided much of the subsequent work. The TCP literature contains several works that apply ML to determine the priority or failure proneness of test cases on the basis of past execution data. (Ramzan, 2026) proposed DD-TCP via gradient-boosted decision trees and historical test-execution features. It achieved a 32.4% APFD improvement and 27.1% lower execution overhead but partly relied on synthetic data and did not compare neural architectures. (Sharma & Chande, 2023) highlighted that neural networks attained the highest fault detection rate of 94% in a comparative ML study. However, their research is limited by reliance on cluster selection and an inability to account for dynamically evolving test cases. (Kumar & Saxena, 2024) applied RF, XGBoost, LR, SVM, and k-NN to ABB Robotics datasets, with RF achieving the highest accuracy of 87%. However, their evaluation depends completely on accuracy, which is inadequate for imbalanced datasets. (Marijan, 2023) carried out a comparative study using GBDT, SVM, ANN, and LRN with extensive hyperparameter tuning, revealing that LRN outperformed other models. (Sheth et al., 2022) compared decision tree, SVM, naïve Bayes, and k-NN methods on five diverse datasets, with naïve Bayes achieving the highest average accuracy of 79.76%.

### 3.3 Reinforcement learning approaches

(Bagherzadeh et al., 2022) proposed a reinforcement learning (RL) approach integrated with feature selection for TCP, which was evaluated on the ABB dataset. (Yaraghi et al., 2023) extended this approach to scalable RL-based TCP in CI contexts. (Spieker et al., 2017) developed the RETECS framework using RL for automatic TCP and selection. (Wu et al., 2019) introduced time window-based RL rewards for the TCP in CI. While RL methods are compelling for online learning scenarios, their reproducibility is often limited by dataset availability and the complexity of reward function design. (Su et al., 2025) proposed ATRL-TCP, which combines reinforcement learning with historical information, error rates, and adaptive attention. An evaluation of four industrial datasets revealed good ranking and generalization performance, but the available source does not report specific APFD/NAPFD values.

### 3.4 Other approaches

(Bertolino et al., 2020) compared learning-to-rank and ranking-to-learn strategies for CI regression testing. (Omri & Sinz, 2022) applied learning-to-rank to TCP. (Elsner et al., 2021) empirically evaluated readily available information for regression test optimization in CIs. (Lima & Regina Vergilio, 2023) also evaluated ranking-to-learn approaches for the TCP. (Felding et al., 2024) used simulated annealing for resource-constrained TCP in an industrial context. (Ghaleb et al., 2019) studied long CIs built on the TravisTorrent dataset. (Cheng et al., 2024) reused TCP on long-running test suites, introducing the LRTS dataset with 21,555 builds from 10 open-source projects. (Tiutin and Vescan, 2026)introduced RoV and FoC criteria for test-case prioritization in software product lines. Across the nine feature models, the new criteria outperformed CoC in 6/9 cases, but the study focused on product-line testing rather than ML-based prioritization in CI environments.

### 3.5 Positioning of the Present Work

The presented AnoMLP-TCP differs from reinforcement learning, attention-based ranking, learning-to-rank, and multimodal TCP for various reasons. One question, however, is whether a feedforward neural network can rival the benefits of the recurrent AnoLSTM-TCP model when temporal aspects have already been chunked into duration, recency, cumulative history, recent verdicts, and the anomaly score. The presented results show that the AnoMLP-TCP produces identical APFD values under the evaluation protocol used in this study as the reproduced LSTM baseline on the same ABB datasets. This is an efficiency result, not a model superiority statement.

## 4 AnoMLP-TCP framework

The proposed framework in Figure 2 performs the data collection, data cleaning and feature engineering, chronological splitting, preprocessing and anoMLP training, inference and prioritization (AnoMLP-TCP), and within-cycle prioritization, evaluating the performance.

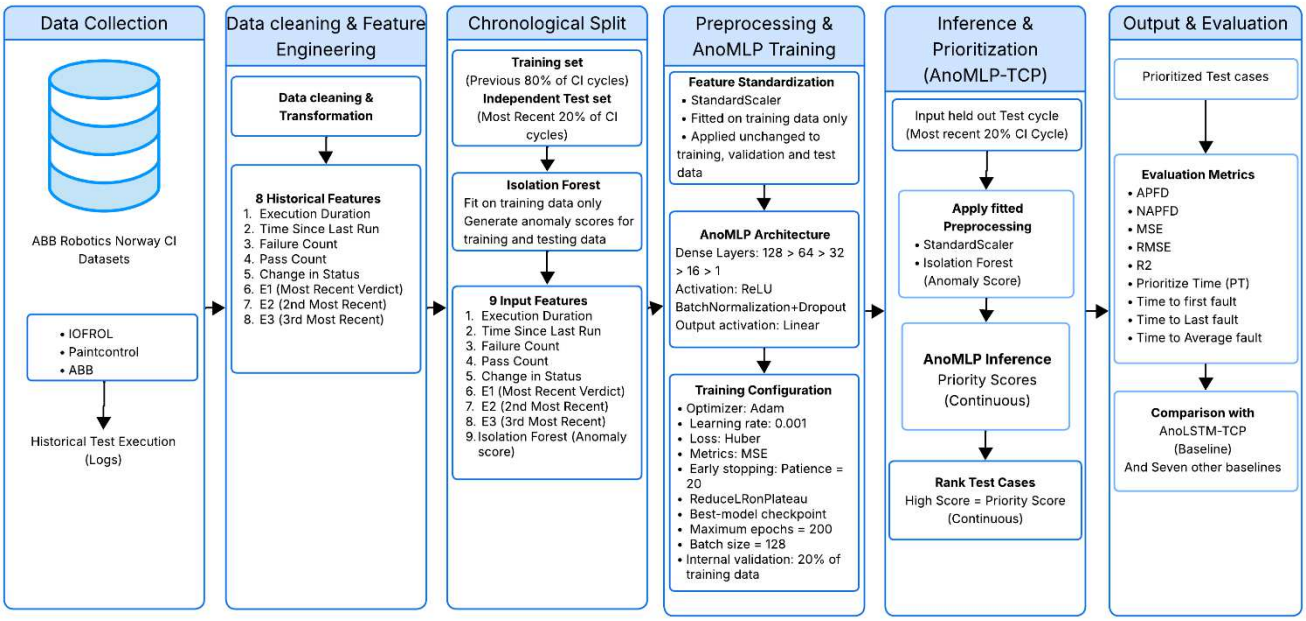



Figure 2: Research Framework

### 4.1 Data collection

The study utilized 3 industrial datasets provided by ABB Robotics Norway, namely, IOFROL, PaintControl, and ABB (tcp). Each dataset entry contains the instance ID, test-case name, execution duration, timestamp of the last execution, list of previous pass/fail outcomes with the current verdict, and CI cycle. In addition, ABB raw data contain an additional CalcPrio field. The data and associated research materials used in this study have been archived in Zenodo to support reproducibility https://doi.org/10.5281/zenodo.23139073

Table 1. Datasets Summary

| | Dataset | Test Cases | CI Cycles | Verdicts | Failure Rate (%) |
|---|---|---|---|---|---|
| **1** | IOFROL | 1941 | 320 | 32260 | 28.79 |
| **2** | PaintControl | 89 | 352 | 25594 | 19.36 |
| **3** | ABB | 604 | 106 | 64024 | 21.36 |

Table 2 illustrates the raw structure common to all three datasets prior to preprocessing, using representative example rows.

Table 2a. Dataset Structure (IOFROL)

| Id | Name | Duration | LastRun | LastResults | Verdict | Cycle |
|---|---|---|---|---|---|---|
| 1 | 78082 | 57262 | 2/13/2015 16:13 | [] | 1 | 1 |
| 2 | 78083 | 57145 | 2/13/2015 16:13 | [] | 1 | 1 |
| 3 | 78084 | 56861 | 2/13/2015 16:13 | [] | 1 | 1 |
| 4 | 78085 | 55332 | 2/13/2015 16:13 | [] | 1 | 1 |
| 5 | 78086 | 59120 | 2/13/2015 16:13 | [] | 1 | 1 |

Table 2b: Dataset Structure (PaintControl)

| Id | Name | Duration | LastRun | LastResults | Verdict | Cycle |
|---|---|---|---|---|---|---|
| 1 | 112180 | 1000 | 1/12/2016 9:17 | [1] | 0 | 1 |
| 2 | 112180 | 1000 | 1/12/2016 9:39 | [0, 1] | 0 | 1 |
| 3 | 112175 | 39050 | 1/12/2016 10:32 | [0] | 0 | 1 |
| 4 | 112180 | 1000 | 1/12/2016 10:34 | [0, 0, 1] | 0 | 1 |
| 5 | 112284 | 47131 | 1/12/2016 10:38 | [0] | 0 | 1 |

Table 2c: Dataset Structure (ABB)

| Id | Name | Duration | CalcPrio | LastRun | LastResults | Verdict | Cycle |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 2.317 | 0 | 1/13/2020 0:00 | [] | 0 | 1 |
| 400 | 400 | 2.533 | 0 | 1/13/2020 0:00 | [] | 0 | 1 |
| 401 | 401 | 4.367 | 0 | 1/13/2020 0:00 | [] | 0 | 1 |
| 402 | 402 | 1.1 | 0 | 1/13/2020 0:00 | [] | 0 | 1 |
| 403 | 403 | 0.5 | 0 | 1/13/2020 0:00 | [] | 0 | 1 |

### 4.2 Preprocessing and causal ordering

We cleaned the datasets by removing all duplicate entries. Any records that have missing or invalid values for Duration, Verdict, LastRun, or Cycle are excluded. We also convert the LastRun field into a datetime format. After that, we sort the remaining records by cycle and LastRun within each cycle. In this way, we can compute the historical features in a causal manner, relying only on the information that was available before the current test execution.

### 4.3 Feature Engineering and Anomaly Detection

The following eight historical features are engineered from the dataset: DurationFeature, TimeSinceLastRun, FailureCount, PassCount, Change_in_status, E1, E2, and E3. DurationFeature is the length of the current run. TimeSinceLastRun is the time between the previous run and the current run. FailureCount and PassCount indicate the total number of failed and passed runs before the current run, respectively. Change_in_status marks whether the verdict of the most recent prior execution differed from the one before it; like all other historical features, it is computed strictly from verdicts recorded before the current execution and never uses the current execution's own Verdict, which remains the held-out prediction target throughout. E1--E3 store the last three verdicts before the current run. Finally, we fitted the isolation forest using only the training cycles. Then, we use the fitted detector to generate anomaly scores for both the training cycle and the held-out test cycle. We then added the anomaly score to the eight engineered historical features, resulting in a total of nine input variables for AnoMLP-TCP.

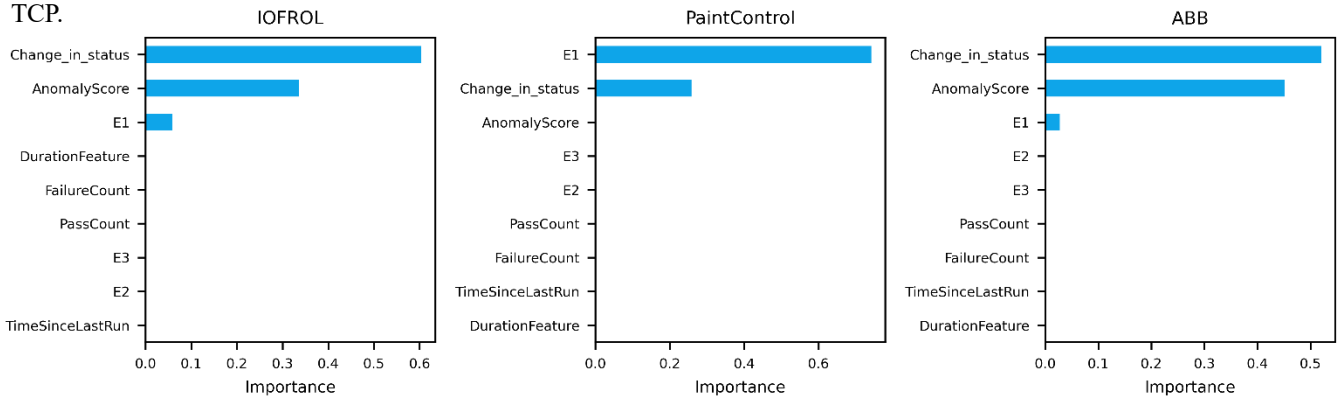


Figure 3: Feature importance

### 4.4 Chronological evaluation

We set aside 20% of the latest CI cycles as an independent test set, using only the earlier cycles for model development. After we split the data chronologically into training and testing sets, we further divide the training data with a 20% validation split. This validation subset is solely for tracking validation loss, implementing early stopping, adjusting the learning rate, and saving model checkpoints. The independent test cycles are kept completely separate until we are ready for the final evaluation. To avoid leaking information from the validation subset into feature scaling, we fit StandardScaler only on the earliest 80% of the training cycles (the model-fitting subset that excludes the validation cycles), and we apply the fitted scaler unchanged to the validation subset and to the held-out test data. This chronological evaluation approach helps minimize the risk of temporal leakage that could occur with a random row-level split that mixes observations from different points in the CI timeline.

### 4.5 AnoMLP-TCP Architecture

While we were reimplementing the LSTM-based baselines, we stumbled upon the dying-ReLU problem. The LSTMTCP and AnoLSTM-TCP baselines were initially described in their papers as using a ReLU activation function in the output layer. However, during our attempts to reproduce their results, we faced several serious training issues. For certain datasets, even after testing thousands of instances, the models we trained only managed to produce between one and five distinct priority values. We traced this odd behavior back to the ReLU output activation, which clips any negative pre-activation values to zero, effectively blocking gradient flow through those output units. This issue appeared consistently across multiple training runs with different random seeds, so it clearly was not just a fluke due to a specific initialization. To address this, we decided to switch the output activation of the AnoLSTM-TCP and LSTMTCP baselines from ReLU to linear, aligning it with the output formulation used by AnoMLP-TCP. This change allowed us to keep the negative pre-activation values and ensured that gradients could flow through the output unit during training.

For AnoLSTM-TCP, we kicked things off by reimplementing the baseline on the basis of the architecture and settings detailed by (Choudhury et al., 2025). Their reported architecture features three LSTM layers, each with 50 units and ReLU activation, along with a dropout rate of 0.2 applied after every LSTM layer. Following these layers, there are three dense layers with 32, 16, and 1 unit, respectively. The model is optimized via the Adam optimizer with a learning rate of 0.001, and it employs the mean squared error (MSE) as the loss function. To double-check our baseline implementation, we reached out to the corresponding author of (Choudhury et al., 2025) and received the original AnoLSTM-TCP implementation. Upon reviewing this implementation, we noticed that the final dense layer does not explicitly define an activation function. As a result, Keras defaults to linear activation. This reveals a mismatch between the published architectural description, which indicates ReLU for the final layer, and the implementation provided by the original authors, which actually uses a linear output.

On the basis of this evidence, the AnoLSTM-TCP baseline evaluated in the present study uses a linear output layer. This is consistent with the implementation obtained from the original authors and with the empirical findings from our independent reproduction, in which the ReLU output produced severe loss of priority score resolution across multiple training runs. Both AnoMLP-TCP and the reproduced AnoLSTM-TCP therefore produce continuous priority scores. The differences in priority score resolution observed between the two models, discussed later in Section 6.2, are consequently interpreted in relation to their respective network architectures and training dynamics rather than as evidence of an output-layer failure in either model.

### 4.6 Algorithm to Generate the AnoMLP-Based TCP Model

**Algorithm:** Developing AnoMLP for Test Case Prioritization

Input: Features engineering and Anomaly score of the test cases

Output: Predicted priority scores and ranked test cases

Step 1: Begin

Step 2: Import necessary packages (NumPy, pandas, scikit-learn, TensorFlow/Keras)

Step 3: Load the IOFROL, PaintControl, and ABB datasets

Step 4: Data Cleaning (Removing Duplicates, Type Conversion and Drop malformations/Missing Record), Sort in chronological order of Cycle and LastRun

Step 5: Construct causal historical features using only information available before the current test execution: DurationFeature, TimeSinceLastRun, FailureCount, PassCount, Change_in_status, E1, E2 and E3.

Step 6: Chronologically split the datasets with the most recent 20% CI cycles as independent test sets and use the preceding cycles for model development.

Step 7: Fit the Isolation Forest using only the model-fitting cycles defined in Step 8.1. Use the fitted detector to generate anomaly scores for the model-fitting, validation, and independent test cycles.

Step 8: Append the anomaly score to the eight engineered historical features to obtain nine model inputs.

Step 8.1: From the chronological model-development portion, reserve the most recent 20% of development cycles as an internal validation subset. The preceding 80% of development cycles are used for model fitting. The StandardScaler is fitted only on this model-fitting subset and then applied unchanged to the validation subset and the independent test set.

Step 8.2: Add four dense layers with 128, 64, 32, and 16 units. The activation function = ReLU.

Step 8.3: Batch normalization on the first three dense layers

Step 8.4: Dropout regularization is applied at rates of 0.30 after the first dense layer and 0.20 after the second dense layer.

Step 8.5: Add one dense output layer with units = 1 and activation function = linear

Step 9: Compile the model

Step 9.1: Optimizer = Adam, Learning_rate = 0.001, Loss = Huber, Metric = MSE

Step 10: Train the model

Step 10.1: Train the AnoMLP-TCP model using the training subset defined in Step 8.1 and monitor its performance on the previously defined internal validation subset.

Step 10.2: Early stopping: Monitor = Val_Loss, Patience = 20, Restore_best_weights = True

Step 10.3: ReduceLROnPlateau: Monitor = Val_Loss, Factor = 0.5, Patience = 8, Min_LR = $1\times10^{-6}$. ModelCheckpoint: Monitor = Val_Loss, Save_best_only = True

Step 10.4: Epoch = 200, Batch_Size = 128

Step 11: Predict continuous priority scores for the held-out test cases and rank test cases in descending order within each CI cycle.

Step 12: Evaluate the ranked test suites cycle-wise using APFD and NAPFD, and evaluate predicted priority scores using MSE, RMSE and $R^2$. Measure fault-detection times (TF, TL, TA) and prioritization time (PT), then aggregate results across valid held-out CI cycles.

Step 13: Terminate

## 5 Experiment Settings

### 5.1 Problem Formulation

Let the Test site be defined by:

$$T = \{t_1, t_2, \dots, t_n\} \tag{1}$$

where n is the total number of Test cases, and

Let the m faults that can be revealed by the Test suite T be defined as:

$$F = \{f_1, f_2, \dots, f_m\} \tag{2}$$

Test case prioritization (TCP) targets the ordering of T that maximises the rate at which faults are detected during execution.

Prioritization problem can be formally expressed as finding a permutation $\pi^*$ that maximises a prioritization effectiveness function $E(\cdot)$:

$$\pi^* = \arg\max_{\pi \in \Pi} E\big(\pi(T)\big) \tag{3}$$

Where Π denotes the set of all $n!$ Possible permutations of T, and $E(\pi(T))$ represents the effectiveness of the resulting test ordering. In our study, $E(\cdot)$ is primarily evaluated using the Average percentage of fault detected (APFD), as defined in section 5.2.1.

In the proposed earning-based approach, the ordering is obtained indirectly through a continuous priority score. For each test case $t_i$, the trained AnoMLP-TCP model $f_\theta$ produces a priority score:

$$P_i = f_\theta(X_i) \tag{4}$$

where:

$X_i$ represents the nine-dimensional feature vector;

$f_\theta$ represents the trained AnoMLP-TCP model; and

$P_i$ represents the predicted continuous priority score.

$X_i$ is the nine-dimensional feature vector comprising DurationFeature, TimeSinceLastRun, FailureCount, PassCount, Change_in_status, E1, E2, E3, and anomaly score. The test cases are therefore sorted in descending order of $P_i$, with the resulting ordering giving the prioritized test suite. Thus, rather than explicitly searching the $n!$ Possible permutations, AnoMLP-TCP learns a scoring function whose outputs gives a test-case ordering. The objective is to produce an ordering that put fault-revealing test cases as early as early as possible in the execution sequence.

**5.2 Experimental Metrics**

*5.2.1 Average percentage of fault detected (APFD)*

It calculates the rate at which faults are detected in the process of testing cases execution in a particular order. The value of APFD lies between 0 and 1.

$$\text{APFD} = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{n \cdot m} + \frac{1}{2n} \tag{5}$$

where:

$n$ = number of test cases

$m$ = number of faults

$TF_i$ = position of the first test case that reveals fault$i$

*5.2.2 Normalized APFD (NAPFD)*

The NAPFD is an extension of the APFD that takes into account the percentage of faults that were detected by the prioritized set.

$$\text{NAPFD} = p - \frac{TF_1 + TF_2 + \cdots + TF_m}{n \cdot m} + \frac{p}{2n} \tag{6}$$

where:

$p$ = ratio of faults detected by the prioritized suite to total faults

*5.2.3 Mean Squared Error (MSE)*

This metric measures the quality of the regression model on the basis of the mean of the squared differences between the observed and predicted values.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{7}$$

*5.2.4 $R^2$ score*

The coefficient of determination, $R^2$, indicates how much of the variation in the actual target values is explained by our model compared with a simple prediction that uses only the mean. It can reach a high value of 1, but it can also dip into a negative territory if the model's predictions are worse than just predicting the mean.

*5.2.5 Root mean square error (RMSE)*

It is the root of the mean squared error. It can be expressed as

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2} \tag{8}$$

*5.2.6 Time for First Fault Detection (TF)*

The time taken to detect the first fault in the prioritized test suite is assessed. Given detection times $T = \{T,(P_1),T,(P_2)\ldots T(P_m)\}$,

It is expressed as$TF = \min(T)$

*5.2.7 Time to detect the last fault (TL)*

This is the time required to detect the last fault. It is given as$TL = \max(T)$

*5.2.8 Time to detect average fault (TA)*

The average detection time across all faults is computed. This is mathematically defined as $TA = \frac{1}{m}\sum_{i=1}^{m} T\,(P_i)$

*5.2.9 Time to Prioritize (PT)*

The total execution time (ranking time) of the prioritization algorithm is measured. Unlike the other reported metrics, PT is a wall-clock measurement and is therefore sensitive to CPU scheduling, system load, and caching effects on the host machine; small run-to-run variation in PT is expected even when the underlying model and its predictions are fully deterministic

*5.2.10 The evaluation metrics used in the study*

APFD is defined over [0, 1], with higher values indicating earlier fault detection. This metric is calculated for each held-out cycle independently, and then the final average is taken over those cycles. Held-out cycles, which contain no failures or failures in all test cases, are omitted from the APFD/NAPFD calculation since the metric yields no informative response in these cases.

NAPFD extends the analysis over partial execution, i.e., when it is possible to detect the fraction of faults within the allocated budget and not all of the tests. When 100% test execution is performed per cycle, NAPFD = APFD.

MSE, RMSE, and $R^2$ serve as regression diagnostics by evaluating the continuous priority scores generated by AnoMLP-TCP in relation to the binary Verdict target. Importantly, these metrics should be understood differently than APFD and NAPFD, which focus directly on measuring how effective the resulting test-case ranking is. TF, TL, and TA are defined over the cumulative time it takes to execute the tests, which results in faults. The PT is measured as the wall-clock time required to score and rank the held-out tests.

**5.3 System requirements**

We implemented the experiments via Python 3.9.12, TensorFlow 2.10.0, and scikit-learn within Jupyter Notebooks in a conda 4.12.0 environment running on a computer with an Intel core i3 (12th generation Intel(R) Core (TM) i3--10110U CPU @ 2.10 GHz processor), 8.00 GB of RAM, a 256 GB SSD and a 64-bit operating system (Windows 11).

**5.4 Research questions**

RQ1. What prioritization effectiveness and predictive-error performance does AnoMLP-TCP achieve across the three evaluated datasets?

RQ2. How does AnoMLP-TCP compare with the implemented baseline, AnoLSTM-TCP, and other "competing" TCP techniques when all the techniques are evaluated via the common protocol defined in this paper?

RQ3. How much computational time savings does AnoMLP-TCP provide relative to the baseline technique, AnoLSTM-TCP?

## 6 Results and Discussion

### 6.1. RQ1: Effectiveness and regression-based priority-score prediction of AnoMLP-TCP

Table 3 presents the top five test cases generated by AnoMLP-TCP for each dataset. These examples show that the predicted priority is not usually based on just one feature. For each test case, AnoMLP-TCP produces a continuous predicted priority score (CalcPrio) from the nine engineered input features. Test cases are ranked in descending order of this predicted priority score, with Rank 1 assigned to the test case having the highest predicted priority.

Table 4 presents the effectiveness and regression-based priority-score prediction performance of AnoMLP-TCP, where PaintControl achieves the highest APFD of 0.959, followed by ABB of 0.893 and IOFROL of 0.821. The full-suite NAPFD values are identical to the APFD values. The $R^2$ values are also very high. However, these regression measures should be taken as diagnostics rather than as evidence; they are not themselves measures of the explanatory quality of the ranking.

Table 3a: Prioritized Test Cases Based on the AnoMLP-TCP Approach - IOFROL

| Id | Duration Feature | TimeSinceLastRun | FailureCount | PassCount | Change_in_status | E1 | E2 | E3 | AnomalyScore | Verdict | CalcPrio | Ranking |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 26781 | 55977 | 954 | 7483 | 19297 | 1 | 0 | 0 | 0 | 0.9246 | 1 | 1.1897 | 1 |
| 25335 | 241239 | 265.05 | 7051 | 18283 | 1 | 0 | 0 | 0 | 0.984 | 1 | 1.0406 | 2 |
| 25253 | 74888 | 144.1333 | 7036 | 18216 | 1 | 0 | 1 | 0 | 0.9518 | 1 | 1.0255 | 3 |
| 31317 | 70708 | 73.8 | 8819 | 22497 | 0 | 1 | 1 | 1 | 0.851 | 1 | 1.0143 | 4 |
| 31307 | 102730 | 68.3667 | 8811 | 22495 | 1 | 0 | 1 | 1 | 0.977 | 1 | 1.013 | 5 |

Table 3b: Prioritized Test Cases Based on the AnoMLP-TCP Approach - PaintControl

| Id | Duration Feature | TimeSinceLastRun | FailureCount | PassCount | Change_in_status | E1 | E2 | E3 | AnomalyScore | Verdict | CalcPrio | Ranking |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 23959 | 29406 | 34.3167 | 4850 | 19108 | 1 | 0 | 0 | 0 | 0.921 | 1 | 1.145 | 1 |
| 23467 | 29406 | 44.65 | 4716 | 18750 | 1 | 0 | 0 | 0 | 0.9309 | 1 | 1.1274 | 2 |
| 24128 | 29406 | 21.2333 | 4858 | 19269 | 1 | 0 | 0 | 0 | 0.9145 | 1 | 1.0801 | 3 |
| 23125 | 29406 | 21.1333 | 4698 | 18426 | 1 | 0 | 0 | 0 | 0.91 | 1 | 1.0799 | 4 |
| 23631 | 29406 | 20.75 | 4720 | 18910 | 1 | 0 | 0 | 0 | 0.8984 | 1 | 1.077 | 5 |

Table 3c: Prioritized Test Cases Based on the AnoMLP-TCP Approach - ABB

| Id | DurationFeature | TimeSinceLastRun | FailureCount | PassCount | Change_in_status | E1 | E2 | E3 | AnomalyScore | Verdict | CalcPrio | Ranking |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 53997 | 5.017 | 0 | 11571 | 42671 | 0 | 1 | 1 | 0 | 0.8329 | 1 | 0.9975 | 1 |
| 52789 | 5.017 | 0 | 11312 | 41723 | 0 | 1 | 1 | 0 | 0.8329 | 1 | 0.9975 | 2 |
| 53393 | 5.017 | 0 | 11441 | 42196 | 0 | 1 | 1 | 0 | 0.8329 | 1 | 0.9975 | 3 |
| 52185 | 5.017 | 0 | 11184 | 41246 | 0 | 1 | 1 | 0 | 0.8329 | 1 | 0.9975 | 4 |
| 51581 | 5.017 | 0 | 11054 | 40772 | 0 | 1 | 1 | 0 | 0.8329 | 1 | 0.9975 | 5 |

Table 4 shows the prioritization times for the proposed model and its competitors on the heldout data for the three training sets, using the final AnoMLP configuration described in Section 4.5. These results are 0.995 s for IOFROL, 0.764 s for PaintControl, and 1.293 s for ABB. These results indicate that AnoMLP-TCP scores and ranks the held-out cases in under 1.3 seconds across the three evaluated datasets. Table 4 also reports the AnoMLP-TCP results for the three datasets in terms of other regression accuracies (MSE and RMSE) and time features (TF, TL, TA, PT). In Figure 4, the training and validation loss curves for AnoMLP-TCP on each dataset are shown. The consistency of the convergence of AnoMLP was maintained across three ABB Robotics datasets: IOFROL, PaintControl, and ABB. At epoch 1, when the model was not trained, Huber losses reached 0.0752, 0.1091, and 0.0614, respectively. Over the training period, the losses decreased to $6.50\times10^{-5}$, $2.07\times10^{-4}$, and $8.61\times10^{-6}$, i.e., 3, 2.7, and 4 orders of magnitude, respectively, for IOFROL, PaintControl, and ABB. Loss function drops were characterized by the standard two-stage procedure fast reduction over the first six epochs (0.075--0.004 for IOFROL,

0.109--0.005 for PaintControl, and 0.061--0.001 for ABB), followed by a slow refinement phase consistent with stable convergence during training. The validation loss remained below or close to the training loss for most epochs, with some fluctuations across the training period but not during testing. The adaptive learning rate policy (ReduceLROnPlateau) reduced the starting learning rate of $1\times10^{-3}$ to $7.8\times10^{-6}$ (IOFROL), $1.25\times10^{-4}$ (PaintControl), and $1.95\times10^{-6}$ (ABB) and produced stepwise descents to new plateaus after each halving of the learning rate. The fluctuations we noticed occurred close to the loss floor and did not line up with any lasting increases in loss. Differences among datasets were evident. First, IOFROL stopped early at epoch 67, followed by PaintControl at epoch 75 and ABB at epoch 54. Second, ABB's validation loss oscillated in a narrow, roughly stable range of $5\times10^{-6}$ to $2\times10^{-5}$ in its last 20 epochs, as opposed to continuing to monotonically decrease, suggesting minibatch noise at the loss floor rather than divergence. The overall decrease in training loss, the consistency between training and validation loss trends, and the stability under learning-rate decay showed stable training behavior on the three ABB Robotics datasets.

Table 4. Model Evaluation Results on the Three Datasets

| Datasets | MSE | R2 | RMSE | APFD | NAPFD | TF(s) | TL(s) | TA(s) | PT(s) |
|---|---|---|---|---|---|---|---|---|---|
| IOFROL | 0.0001 | 0.9994 | 0.0112 | 0.821 | 0.821 | 228380.9 | 6012996 | 3447173 | 0.995 |
| PaintControl | 0.0002 | 0.9973 | 0.0133 | 0.959 | 0.959 | 37759.36 | 240458.1 | 133671.1 | 0.764 |
| ABB | 0 | 0.9999 | 0.0039 | 0.893 | 0.893 | 276.06 | 56382.4 | 40262.66 | 1.293 |

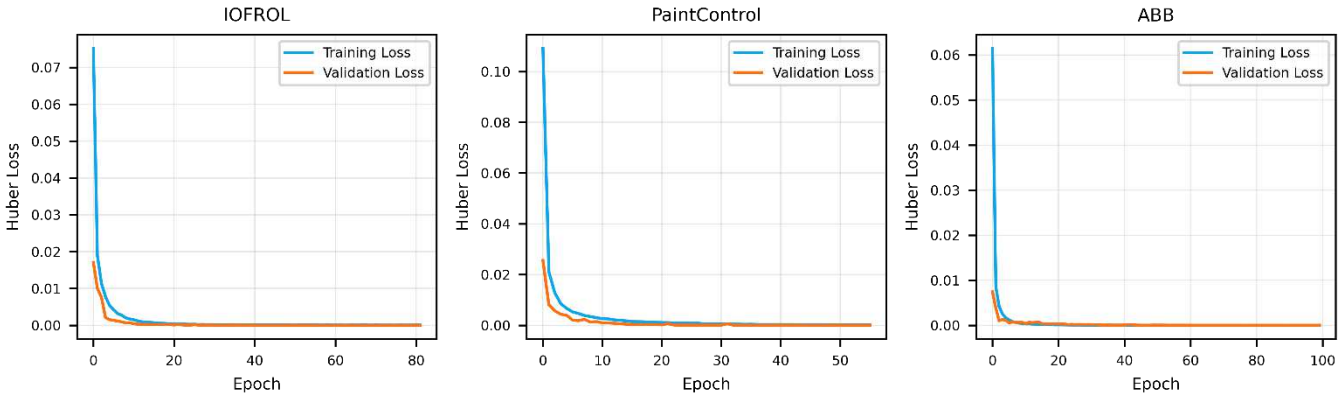



Figure 4. Training and Validation Loss Curves

### 6.2 RQ2: Comparison with Independently Implemented Baselines

Table 5 shows an experimental baseline comparison. Every value shown in the table is our reimplementation of the method in question and is evaluated with the same general data preparation and cycle-based protocol.

***Test Cycles***: Denotes the total number of chronological test cycles contained in the held-out test period.

***Valid Cycles*:** Denotes the subset of these test cycles retained for APFD and NAPFD computation after applying the predefined validity criteria. Specifically, cycles containing no failed test cases or in which all test cases fail are excluded because they do not provide meaningful discrimination for fault-oriented prioritization evaluation.

Table 5: Effectiveness - APFD/NAPFD (same cycle-based split, nine methods)

| Method | Dataset | APFD | APFD_std | NAPFD | TestCycles | ValidCycles |
|---|---|---|---|---|---|---|
| RETECS-N | ABB | 0.893 | 0 | 0.893 | 21 | 21 |
| DeepGini | ABB | 0.893 | 0 | 0.893 | 21 | 21 |
| LSTMTCP | ABB | 0.893 | 0 | 0.893 | 21 | 21 |
| HyLSTMTCP | ABB | 0.893 | 0 | 0.893 | 21 | 21 |
| AnoLSTM | ABB | 0.893 | 0 | 0.893 | 21 | 21 |
| AnoMLP | ABB | 0.893 | 0 | 0.893 | 21 | 21 |
| DeepOrder | ABB | 0.745 | 0.006 | 0.745 | 21 | 21 |
| ROCKET | ABB | 0.688 | 0.003 | 0.688 | 21 | 21 |
| Random | ABB | 0.499 | 0.023 | 0.499 | 21 | 21 |
| RETECS-N | IOFROL | 0.821 | 0.127 | 0.821 | 64 | 43 |

| | | | | | | |
|---|---|---|---|---|---|---|
| DeepGini | IOFROL | 0.821 | 0.127 | 0.821 | 64 | 43 |
| LSTMTCP | IOFROL | 0.821 | 0.127 | 0.821 | 64 | 43 |
| HyLSTMTCP | IOFROL | 0.821 | 0.127 | 0.821 | 64 | 43 |
| AnoLSTM | IOFROL | 0.821 | 0.127 | 0.821 | 64 | 43 |
| AnoMLP | IOFROL | 0.821 | 0.127 | 0.821 | 64 | 43 |
| DeepOrder | IOFROL | 0.578 | 0.157 | 0.578 | 64 | 43 |
| ROCKET | IOFROL | 0.576 | 0.2 | 0.576 | 64 | 43 |
| Random | IOFROL | 0.503 | 0.088 | 0.503 | 64 | 43 |
| RETECS-N | PaintControl | 0.959 | 0.057 | 0.959 | 70 | 52 |
| DeepGini | PaintControl | 0.959 | 0.057 | 0.959 | 70 | 52 |
| LSTMTCP | PaintControl | 0.959 | 0.057 | 0.959 | 70 | 52 |
| HyLSTMTCP | PaintControl | 0.959 | 0.057 | 0.959 | 70 | 52 |
| AnoLSTM | PaintControl | 0.959 | 0.057 | 0.959 | 70 | 52 |
| AnoMLP | PaintControl | 0.959 | 0.057 | 0.959 | 70 | 52 |
| DeepOrder | PaintControl | 0.721 | 0.191 | 0.721 | 70 | 52 |
| ROCKET | PaintControl | 0.576 | 0.241 | 0.576 | 70 | 52 |
| Random | PaintControl | 0.473 | 0.189 | 0.473 | 70 | 52 |

ValidCycles refers to the number of held-out CI cycles that include at least one failing test case but not all of them. The metrics APFD and NAPFD are calculated solely for these informative cycles. The figures presented in Table 5 are from the independent implementations we used in this study and were not taken from the original research.

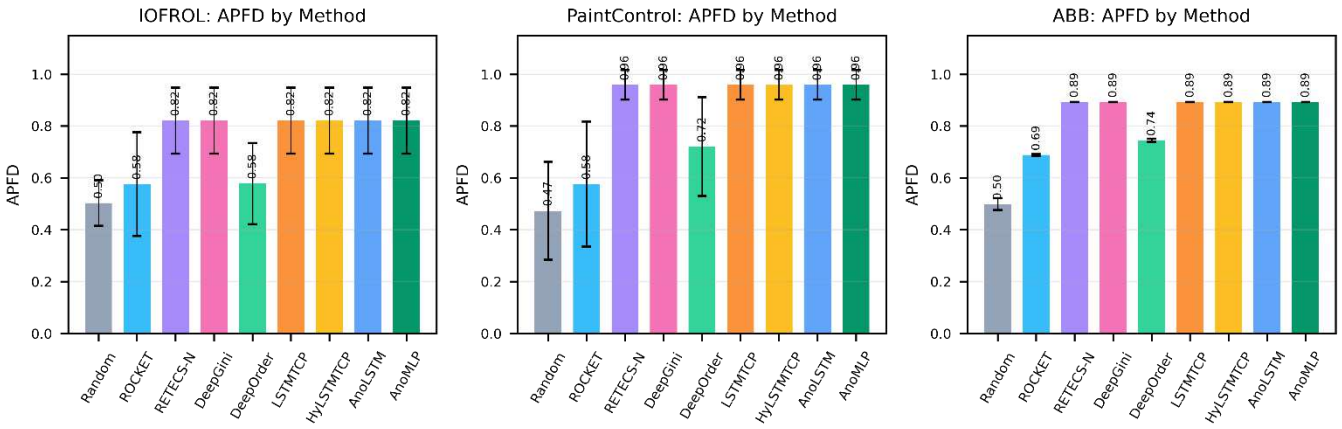



Figure 5. APFD comparison across methods

The comparison shows that AnoMLP-TCP jointly achieves the highest APFD values with five baseline methods on all three datasets, whereas five of the eight baseline implementations (RETECS-N, DeepGini, LSTMTCP, HyLSTMTCP, and the reimplemented AnoLSTM-TCP) achieve the same per-cycle APFD as AnoMLP on each dataset. A diagnostic analysis in Table 6 shows that this equality is not an implementation artifact but rather a direct, provable consequence of the APFD formula: for every cycle and dataset evaluated, these five methods rank the complete set of fault-revealing test cases within the first m ranked positions, where m is the number of faults in that cycle, just like AnoMLP-TCP does. The APFD formula in Section 5.2.1 sums only the rank positions occupied by fault-revealing tests, irrespective of their internal order, so that the APFD is maximized when all the fault-revealing test cases appear in the first *m* positions of the prioritized sequence, where *m* is the number of fault-revealing tests. Thus, various internal orderings of those fault-revealing tests do not impact the APFD. The results per cycle in Table 6 reveal that for each of the five methods, the APFD difference from AnoMLP-TCP was consistently zero across all the evaluated cycles. Since all the paired differences were zero, we could not estimate the Wilcoxon signed-rank test. Therefore, the Wilcoxon signed-rank test was not estimable for any of these methods. The remaining three baselines (DeepOrder, ROCKET, and Random) achieve perfect failure-set separation in at most 2.3% of cycles and differ from AnoMLP-TCP with a statistically significant difference ($p < 0.05$) on all datasets. It is important to see this as a ceiling effect instead of just a matter of architectural equivalence. In these datasets, features such as Change_in_status and E1 seem to offer significant discriminative power, enabling well-designed models to effectively differentiate between fault-revealing tests and the others. As a result, APFD might struggle to differentiate between methods that successfully perform this separation.

Further analysis through a comparison of raw priority scores provides an interesting perspective. As explained in Section 4.5, the implementation of both AnoMLP-TCP and the newly implemented AnoLSTM-TCP includes a linear output layer, which means that they do not enforce the non-negative truncation constraint that the ReLU introduces. However, the difference between these two approaches in terms of score granularity is substantial compared with the other eight baseline methods (as few as 4 distinct values of prediction for the LSTMTCP and several thousand distinct priority score values for HyLSTMTCP and ROCKET). The large gap in terms of score granularity was not reflected in any gap in terms of APFD; some approaches that generated very few unique scores scored equally to AnoMLP-TCP, whereas some approaches that used thousands of scores were different. This finding is expected on the basis of the mechanism explained above, as the APFD metric cares about nothing but whether fault-revealing tests belong to the right subset of highest ranks.

**Table 6**: Failing-Set Separation Diagnostic and Wilcoxon Significance (AnoMLP-TCP vs Each Baseline)

| Dataset | Method | Valid Cycles | Failing-Set Separation (%) | Wilcoxon p | Significant (P<0.05) |
|---|---|---|---|---|---|
| ABB | RETECS-N | 21 | 100 | n/a (no variance) | --- |
| ABB | DeepGini | 21 | 100 | n/a (no variance) | --- |
| ABB | LSTMTCP | 21 | 100 | n/a (no variance) | --- |
| ABB | HyLSTMTCP | 21 | 100 | n/a (no variance) | --- |
| ABB | AnoLSTM | 21 | 100 | n/a (no variance) | --- |
| ABB | DeepOrder | 21 | 0 | <0.001 | Yes |
| ABB | ROCKET | 21 | 0 | <0.001 | Yes |
| ABB | Random | 21 | 0 | <0.001 | Yes |
| IOFROL | RETECS-N | 43 | 100 | n/a (no variance) | --- |
| IOFROL | DeepGini | 43 | 100 | n/a (no variance) | --- |
| IOFROL | LSTMTCP | 43 | 100 | n/a (no variance) | --- |
| IOFROL | HyLSTMTCP | 43 | 100 | n/a (no variance) | --- |
| IOFROL | AnoLSTM | 43 | 100 | n/a (no variance) | --- |
| IOFROL | DeepOrder | 43 | 0 | <0.001 | Yes |
| IOFROL | ROCKET | 43 | 2.3 | <0.001 | Yes |
| IOFROL | Random | 43 | 0 | <0.001 | Yes |
| PaintControl | RETECS-N | 52 | 100 | n/a (no variance) | --- |
| PaintControl | DeepGini | 52 | 100 | n/a (no variance) | --- |
| PaintControl | LSTMTCP | 52 | 100 | n/a (no variance) | --- |
| PaintControl | HyLSTMTCP | 52 | 100 | n/a (no variance) | --- |
| PaintControl | AnoLSTM | 52 | 100 | n/a (no variance) | --- |
| PaintControl | DeepOrder | 52 | 0 | <0.001 | Yes |
| PaintControl | ROCKET | 52 | 1.9 | <0.001 | Yes |
| PaintControl | Random | 52 | 0 | <0.001 | Yes |

### 6.3 Budget Analysis for Test Selection

Budget analysis is performed to determine whether AnoMLP-TCP and AnoLSTM-TCP are still equivalent when only a subset of each test suite is executed. We consider a full-suite budget of 100% and three partial test-selection budgets of 50%, 70% and 90%.

The budget results obtained in Table 7 show that the NAPFD is the same for both models for every budget level. When applied to IOFROL, the NAPFD for a 50% budget is 0.795 and increases to 0.819 for 70% and 0.821 for 90% budgets. The full suite value is also 0.821. For PaintControl and ABB, the NAPFD on all tested budgets is the same as the full-suite value. For PaintControl and ABB, the values remain 0.959 and 0.893 throughout the budgets, respectively.

Table 7. AnoMLP-TCP vs. AnoLSTM-TCP NAPFD Under Test-Selection Budgets

| Dataset | Budget | AnoLSTM_NAPFD | AnoMLP_NAPFD | Gap |
|---|---|---|---|---|
| IOFROL | 50% | 0.795 | 0.795 | 0 |
| IOFROL | 70% | 0.819 | 0.819 | 0 |
| IOFROL | 90% | 0.821 | 0.821 | 0 |
| IOFROL | 100% | 0.821 | 0.821 | 0 |
| PaintControl | 50% | 0.959 | 0.959 | 0 |
| PaintControl | 70% | 0.959 | 0.959 | 0 |

| | | | | |
|---|---|---|---|---|
| PaintControl | 90% | 0.959 | 0.959 | 0 |
| PaintControl | 100% | 0.959 | 0.959 | 0 |
| ABB | 50% | 0.893 | 0.893 | 0 |
| ABB | 70% | 0.893 | 0.893 | 0 |
| ABB | 90% | 0.893 | 0.893 | 0 |
| ABB | 100% | 0.893 | 0.893 | 0 |

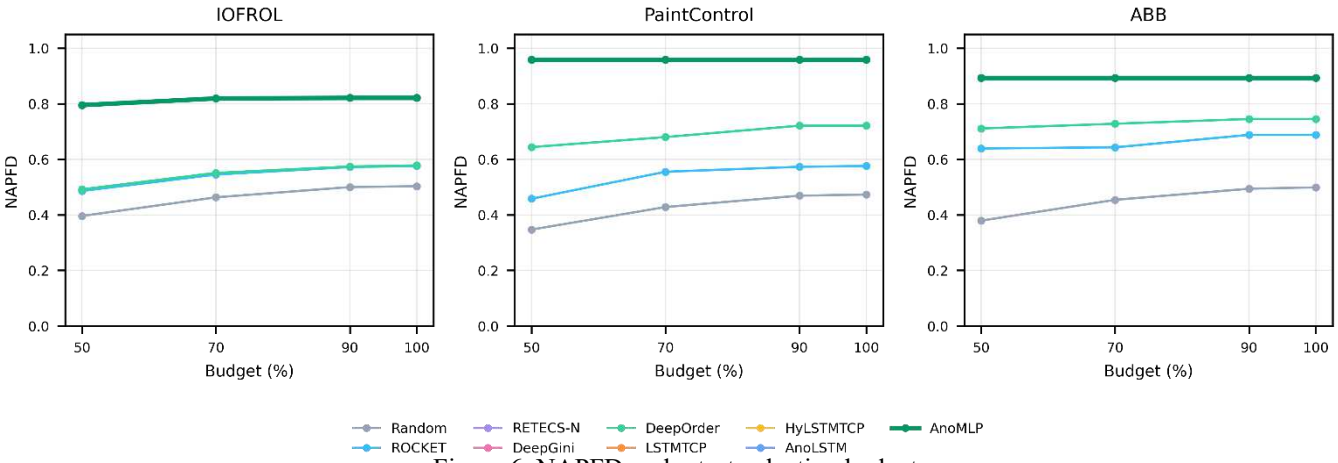



Figure 6. NAPFD under test-selection budgets.

### 6.4 RQ3: Computational cost

Table 8 shows the total parameter count and training time for the AnoMLP-TCP and AnoLSTM-TCP models. The two models were run using the same maximum-epoch and early stopping criteria for the computations. Since the models were assessed in the same computational setting and followed a generally consistent training protocol, the differences we observe in parameter count and training time per epoch mainly highlight variations in model architecture and how they were implemented.

Table 8. Parameter Count and Training Time, AnoMLP-TCP vs. AnoLSTM-TCP

| | Method | Dataset | Total Parameters | Epochs Run | Training Time(s) | Time Per Epoch(s) |
|---|---|---|---|---|---|---|
| **1** | AnoMLP | IOFROL | 13057 | 67 | 48.2 | 0.719 |
| **2** | AnoLSTM | IOFROL | 54577 | 45 | 64.185 | 1.426 |
| **3** | AnoMLP | PaintControl | 13057 | 75 | 43.027 | 0.574 |
| **4** | AnoLSTM | PaintControl | 54577 | 79 | 74.49 | 0.943 |
| **5** | AnoMLP | ABB | 13057 | 54 | 67.982 | 1.259 |
| **6** | AnoLSTM | ABB | 54577 | 30 | 69.711 | 2.324 |

The MLP-based AnoMLP-TCP model uses only 13,057 total parameters, whereas the AnoLSTM-TCP model uses 54,577 total parameters, a reduction in model size of approximately 76.1% in total parameter count. The per-epoch training time for AnoMLP-TCP is lower than that for AnoLSTM-TCP on all three datasets presented in this study: 0.719 s versus 1.426 s on IOFROL, 0.574 s versus 0.943 s on PaintControl, and 1.259 s versus 2.324 s on ABB. These are reductions of approximately 49.6%, 39.1%, and 45.8%, respectively. In Table 8, the AnoMLP-TCP model required more epochs before stopping than did the AnoLSTM-TCP model on IOFROL and ABB but fewer epochs on PaintControl (75 versus 79); however, the total wall-clock training time was lower across all three datasets. The strongest evidence of efficiency in this study therefore comes from the cost per epoch, as well as the total training time.

## 7 Threats to Validity

*Dataset validity*: The three sets are industrial and perhaps valuable, but they are all from ABB Robotics; therefore, they generalize only to that domain, the company, and their CI environments.
*Temporal validity*: Using a chronological split is a better choice than a random row-level split for assessing future CI deployments. This approach ensures that future cycles are not used to make predictions about earlier cycles. However, notably, each dataset employs only a single chronological hold-out boundary. To strengthen the evidence and achieve more reliable

confidence estimates, techniques such as rolling-window evaluation or repeated chronological cross-validation could be beneficial, allowing for multiple temporal splits.

*Baseline validity*: All baseline values in Table 5 are from this study's implementations, which are improved to aid comparability of reported numbers, but independent implementations are not equal to copying source code from the original authors. The AnoLSTM-TCP was first reimplemented on the basis of the algorithm and architectural details that were published. After that, we obtained the original implementation directly from the corresponding author and took a close look to confirm the architectural specifics and any differences in the protocols.

There are two main differences between the evaluation protocol used in this paper and that used by the original authors. The first difference is related to the fact that the original implementation uses a random split of 80:20 row-level (using sklearn.model_selection.train_test_split, seeded), whereas in this work, the split is performed chronologically, cycle-by-cycle, in accordance with the threat of temporal leakage, which the chronological protocol in Section 4.4 is supposed to address. The second difference is that the original implementation calculates two different effectiveness numbers, namely, the single APFD calculated on the basis of ranking all test instances of all CI cycles as a single list and the NAPFD average per cycle, where cycles with zero failures have an average value of 0 (and are not excluded, as in this work) (Section 5.2.2). In this context, the results for AnoLSTM-TCP presented in this work should be interpreted as results obtained within an independent evaluation protocol consistently applied to all nine considered approaches. Some minor differences in the implementations include MinMaxScaler rather than StandardScaler and early-stopping patience of 50 rather than 20 epochs.

*Statistical validity*: This study reports the means and standard deviations for APFD and additionally conducts a paired Wilcoxon signed-rank test on per-cycle APFD between AnoMLP-TCP and each baseline method in Table 6, rather than relying on aggregate values alone. A related, separate threat concerns construct validity: for five of the eight baselines, the per-cycle APFD difference against AnoMLP-TCP was exactly zero in every evaluated cycle, which a follow-up diagnostic traced to a ceiling effect in the APFD metric itself on these three datasets; discriminative features such as Change_in_status and E1 in Figure 3 allow any sufficiently capable model to separate the fault-revealing tests from the rest of the suite essentially perfectly, leaving APFD unable to differentiate further among methods that clear that bar. This means that APFD equivalence between AnoMLP-TCP and AnoLSTM-TCP should not, by itself, be read as evidence of comparable ranking behavior beyond the fault-revealing subset; a metric less prone to saturation on these datasets (e.g., a full Spearman rank correlation over the whole prioritized suite or evaluation on datasets with a higher average fault count per cycle) is a natural direction for future validation.

*Training stability*: All the experiments employ random seeds for NumPy, TensorFlow, and Python's random module while running the code in single thread mode to prevent the drift of nondeterministic multithreaded floating-point reduction; the stability of the results was confirmed by rerunning all the experiments from scratch within the entire notebook. The experiments are not multi-seed experiments, so the results show the behavior of each architecture after being initialized once only; therefore, it may still be possible that the difference between untrained and optimized networks (recurrent vs. feedforward) will change under another seed.

## 8. Conclusion and Future Work

We propose AnoMLP-TCP, an anomaly aware MLP framework aimed at examining the possibility of using a simpler feedforward network to develop an effective TCP framework as an alternative to recurrent deep learning approaches. We aimed to determine whether the use of a recurrent architecture in AnoLSTM-TCP is essential for obtaining effective prioritization or whether comparable results can be achieved with a less complex feedforward architecture. The results from our experiments on the IOFROL, PaintControl, and ABB datasets demonstrate that AnoMLP-TCP achieves prioritization effectiveness comparable to that of the reproduced AnoLSTM-TCP baseline while substantially reducing model complexity and training cost per epoch. Compared with AnoLSTM-TCP, AnoMLP-TCP achieves a remarkable 76.1% reduction in total parameters and has a lower training cost than AnoLSTM-TCP on all the tested datasets. Model size reduction is substantial for CI environments because test prioritization may need to be repeated in the process of continuous software development. It will be possible to train smaller models and, thus, to obtain a more practical alternative to more complex approaches for situations where it is necessary to update models and prioritize tests repeatedly. On the other hand, the results regarding the APFD and NAPFD suggest that in the evaluation framework of this study, simplifying the model did not lead to any noticeable drop in the APFD or NAPFD in the context of the current study. The idea behind introducing AnoMLP-TCP is not to claim that MLP-based approaches are superior to other TCP approaches in general but rather to provide a reference framework for exploring the necessary complexity of an architecture to achieve effective anomaly aware test-case prioritization. AnoMLP-

TCP can be used as a simplified baseline framework in the process of future research on TCP, including reinforcement learning or learning-to-rank approaches. Such comparisons can reveal whether the increased complexity of an approach leads to a meaningful improvement in fault-detection effectiveness. Several avenues remain open for further research. First, it is necessary to extend the experiments to a multi-seed setting in which AnoMLP-TCP and AnoLSTM-TCP are trained several times with different random seeds. This would provide stronger evidence regarding the stability of the obtained results. Second, it is necessary to perform experiments using several training and testing folds chronologically to provide a more accurate evaluation of model performance. In this case, it would be possible to estimate confidence intervals and apply appropriate paired tests to matched CI cycles. Moreover, AnoMLP-TCP should be tested against AnoLSTM-TCP and other TCP methods using the same datasets, the same preprocessing procedures, and the same evaluation protocol.

## Acknowledgments

The contribution given by the faculty staff, both teaching and not teaching in carrying out this research work, is highly acknowledged.

**Authors contribution**

Musa Isyaku contributed to conceptualization, methodology, data curation, software, investigation, formal analysis, and writing of the original draft. Abdulsalam Ya'u Gital contributed to supervision, writing-review and editing. Haruna Chiroma also contributed to supervision, writing, review and editing. All authors read and approved the final manuscript.

**Declaration of competing interest**

The authors declare that they have no competing interests.

**Funding**

The authors did not receive support from any organization for the submitted work.

**Data availability**

The datasets and research materials used to support the findings of this study are publicly available from Zenodo at https://doi.org/10.5281/zenodo.23139073.

**REFERENCES**

Bagherzadeh, M., Kahani, N., & Briand, L. (2022). Reinforcement Learning for Test Case Prioritization. *IEEE Transactions on Software Engineering*, *48*(8), 2836–2856. https://doi.org/10.1109/TSE.2021.3070549

Behera, A., & Abhinna, A. (2025). ScienceDirect ScienceDirect An Effective GRU-Based Deep Learning Method for Test Case Prioritization in Continuous Integration Testing. *Procedia Computer Science*, *258*, 4070–4083. https://doi.org/10.1016/j.procs.2025.04.658

Bertolino, A., Guerriero, A., Miranda, B., Pietrantuono, R., & Russo, S. (2020). Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. *Proceedings - International Conference on Software Engineering*, 1–12. https://doi.org/10.1145/3377811.3380369

Cheng, R., Wang, S., Jabbarvand, R., & Marinov, D. (2024). Revisiting Test-Case Prioritization on Long-Running Test Suites. *ISSTA 2024 - Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 615–627. https://doi.org/10.1145/3650212.3680307

Choudhury, T. K., Behera, M., Dash, S. K., & Kumar, S. (2025). AnoLSTM-A Deep Learning Approach for Test Cases Prioritization Learning Approach Test Cases Prioritization. *Procedia Computer Science*, *258*, 1793–1803. https://doi.org/10.1016/j.procs.2025.04.431

Elsner, D., Hauer, F., Pretschner, A., & Reimer, S. (2021). Empirically evaluating readily available information for regression test optimization in continuous integration. *ISSTA 2021 - Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 491–504. https://doi.org/10.1145/3460319.3464834

Felding, E., Strandberg, P. E., Quttineh, N. H., & Afzal, W. (2024). Resource Constrained Test Case Prioritization with Simulated Annealing in an Industrial Context. *Proceedings of the ACM Symposium on Applied Computing*, 1694–1701. https://doi.org/10.1145/3605098.3635971

Ghaleb, T. A., da Costa, D. A., & Zou, Y. (2019). An empirical study of the long duration of continuous integration builds. *Empirical Software Engineering*, *24*(4), 2102–2139. https://doi.org/10.1007/s10664-019-09695-9

Spieker H., Marijan D., Gotlieb A., Mossige M. (2017). Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration. *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, *April*, 12–22. https://doi.org/https://doi.org/10.1145/3092703.3092709

Romero J. R., Ramírez A., García-Martínez C. (2026). Automated machine learning for test case prioritisation. *Empirical Software Engineering*, *31*(5), 1–41. https://doi.org/10.1007/s10664-026-10862-y

Kalyani, P., Rao, C. P., Goparaju, B., Babu, K. K., & Kandimalla, P. C. R. (2025). BugPrioritizeAI for multimodal test case prioritisation using bug reports, code changes, and test metadata. *Scientific Reports*, *16*(1), 1539. https://doi.org/10.1038/s41598-025-31851-z

Khan, A., Azim, A., & Liscano, R. (2025). ML-Based Test Case Prioritization : A Research and Production Perspective in CI Environments. *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*, 476–486. https://doi.org/10.1109/ICST62969.2025.10989029

Kumar, H., & Saxena, V. (2024). Machine Learning for Test Case Prioritization in Continuous Integration: A Comprehensive Analysis. *International Journal Advanced Networking and Applications*, *17*(02), 6218–6228. https://doi.org/10.35444/ijana.2024.15609

Lima, J. A. do P., & Regina Vergilio, S. (2023). An Evaluation of Ranking-to-Learn Approaches for Test Case Prioritization in Continuous Integration. *Journal of Software Engineering Research and Development*. https://doi.org/10.5753/jserd.2023.2142

Lousada, J., & Ribeiro, M. (2020). Neural Network Embeddings for Test Case Prioritization. *ArXiv E-Prints*, arXiv:2012.10154. https://doi.org/10.48550/arXiv.2012.10154

Marijan, D. (2023). Comparative study of machine learning test case prioritization for continuous integration testing. *Software Quality Journal*, *31*(4), 1415–1438. https://doi.org/10.1007/s11219-023-09646-0

Medhat, N., Moussa, S. M., Badr, N. L., & Tolba, M. F. (2020). A Framework for Continuous Regression and Integration Testing in IoT Systems Based on Deep Learning and Search-Based Techniques. *IEEE Access*, *8*, 215716–215726. https://doi.org/10.1109/ACCESS.2020.3039931

Mustaqeem M, Mustajab S, A. M., & Jeribi F, Alam S, S. M. (2024). (2024). A trustworthy hybrid model for transparent software defect prediction : SPAM-XAI. *PLoS ONE*, *19(7)*(e0307112), 1–33. https://doi.org/10.1371/journal.pone.0307112

Omri, S., & Sinz, C. (2022). Learning to Rank for Test Case Prioritization. *Proceedings - 15th Search-Based Software Testing Workshop, SBST 2022*, 16–24. https://doi.org/10.1145/3526072.3527525

Ramzan, H. A. R. and K. I. and M. A. H. and R. M. M. and S. M. A. and S. (2026). Data-Driven Test Case Prioritization (DD-TCP): A Machine Learning Framework for Intelligent Software Quality Assurance. *Computers, Materials and Continua*, *88*(1). https://doi.org/https://doi.org/10.32604/cmc.2026.077782

Shankar, R., & Sridhar, D. (2024). An Improved Deep Learning Based Test Case Prioritization Using Deep Reinforcement Learning. *International Journal of Intelligent Engineering and Systems*, *17*(1), 771–782. https://doi.org/10.22266/ijies2024.0229.64

Sharif, A., Marijan, D., & Liaaen, M. (2021). DeepOrder: Deep Learning for Test Case Prioritization in Continuous Integration Testing. *Proceedings - 2021 IEEE International Conference on Software Maintenance and Evolution, ICSME 2021*, 525–534. https://doi.org/10.1109/ICSME52107.2021.00053

Sharma, S., & Chande, S. V. (2023). Optimizing test case prioritization using machine learning algorithms. *Journal of Autonomous Intelligence*, *6*(2), 1–16. https://doi.org/10.32629/jai.v6i2.661

Sheth, V., Tripathi, U., & Sharma, A. (2022). A Comparative Analysis of Machine Learning Algorithms for Classification Purpose. *Procedia Computer Science*, *215*, 422–431. https://doi.org/10.1016/j.procs.2022.12.044

Su, Q., & , Xingze Li, Yuming Ren, Ruinan Qiu, C. H. and Y. Y. (2025). Attention Transfer Reinforcement Learning for Test Case Prioritization in Continuous Integration. *A Replication and Extension Study*, *February*, 1–22. https://doi.org/https://doi.org/10.3390/ app15042243

Tiutin, Andrada Georgia and Vescan, A. (2026). New criteria for test case prioritization for software product lines. A replication and extension study. *Automated Software Engineering. Advance Online Publication.*, *33*(3), 1–26. https://doi.org/10.1007/s10515-026-00619-8

Wu, Z., Yang, Y. Y., Li, Z., & Zhao, R. (2019). A time window based reinforcement learning reward for test case prioritization in continuous integration. *ACM International Conference Proceeding Series*, 2–7. https://doi.org/10.1145/3361242.3361258

Yaraghi, A. S., Bagherzadeh, M., Kahani, N., & Briand, L. C. (2023). Scalable and Accurate Test Case Prioritization in Continuous Integration Contexts. *IEEE Transactions on Software Engineering*, *49*(4), 1615–1639. https://doi.org/10.1109/TSE.2022.3184842

Isyaku, M. (2026). *AnoMLP-TCP datasets and research materials*. Zenodo. https://doi.org/10.5281/zenodo.23139073